model. This type of use applies to all accessors. A model accessor adds no additional events.

[0238] An interface accessor of the present invention is an accessor that provides access to implementations of interfaces. While the interface definition provides the details for both static attributes and the required operations to implement an interface, the operations are implemented by model implementations, not interface implementations. Therefore, interface accessors provide access to static attributes from the interface, but require a model instance to perform operations.

[0239] An interface accessor has a one-to-one association relationship with an interface descriptor that is the interface descriptor for which the interface accessor provides access to an implementation. There exists a one-to-one relationship for each descriptor in the interface descriptor to each child accessor in this interface accessor. An interface accessor has a one-to-one association relationship with an interface implementation that is the implementation of the interface. An interface accessor has a zero-to-many aggregation relationship with interface accessors that are wholly included as part of itself. An interface accessor has a zero-to-many aggregation relationship with static attribute accessors that are attribute accessors that do not require an instance in order to be accessed. These accessors allow the value to be retrieved but not changed. An interface accessor has a zero-to-many aggregation relationship with operation accessors that require an instance of a model instance implementing this interface in order to be executed. These accessors provide a mechanism to execute the operation.

[0240] The accessors contained in an interface accessor provide the operations available. An interface accessor adds no additional events.

[0241] A package accessor of the present invention is an accessor that provides access to the various logical groupings of models. The root package accessor may be stored in a well-known location, like the metamodel repository, in order for software applications to be able to find meta-implementation accessors and virtual implementations.

[0242] A package accessor has a one-to-one association relationship with a package descriptor that is the package descriptor for which the package accessor provides access to an implementation. A package accessor has a one-to-one association relationship with a package implementation that is the package implementation that the interface accessor uses to perform the action. A package accessor has a zero-to-many association relationship with metamodel implementations that are the metamodel accessors that are children of this package accessor refer to the metamodel implementations that are children of this package accessor's implementation. A package accessor has a zero-to-many association relationship with package implementations that are the package accessors that are children of this package accessor refer to the package implementations that are children of this package accessor's implementation. A package accessor has a zero-to-many association relationship with package accessors that are child package accessors describing child package implementations of this package accessor's package implementation. A package accessor has a zero-to-many association relationship with model accessors (that are child model accessors describing child model implementations of this package accessor's package implementation. A package accessor has a zero-to-many aggregation relationship with attribute accessors that are the accessors to retrieve other attributes of the package.

[0243] Package accessors are purely descriptive and provide access to other package and model accessors. A package accessor adds no additional events.

[0244] Versions are not accessed. Instead, the version of the accessor is described using a version implementation. This is equivalent to the descriptors in the metamodeling layer holding a version instance describing the version of the descriptor.

[0245] No hint accessors exist. Hints exist only in the modeling layer. Hints capture descriptions necessary for implementations that are not captured elsewhere. Once an implementation exists or is accessed via an accessor, there is no longer a need for hints.

[0246] No role accessors exist. Roles exist only in the modeling layer. In the meta-implementation layer, interfaces serve a similar position.

[0247] In one embodiment, the present invention provides virtual implementations. Where meta-implementation accessors serve to access a real implementation written and compiled in source code, a virtual implementation serves as both the meta-implementation and the "real" implementation. The virtual implementation creates a one-to-one relationship from its assembly of implementations to the descriptors in the meta-implementation layer. Each virtual implementation type works as part of the larger virtual implementation to create assemblies that behave exactly like a real implementation.

[0248] Accessors may use the assemblies of virtual implementations in exactly the same way as a real implementation. The virtual implementation assemblies behave exactly like real implementations. The difference is that virtual implementations are constructed (and optionally modifiable) at runtime. No source code is written, no source code is compiled, and no source code needs to be managed. Given a tool which properly constructs a virtual model implementation assembly from a model descriptor (a one-to-one mapping of each feature descriptor to a feature implementation), virtual implementations require no more effort to create than a fully documented UML diagram.

[0249] A user (a human or other type of user) may use a virtual implementation as the meta-implementation layer in the same way that it uses accessors. Since the meta-implementation serves as a layer of indirection between the user and the implementation, the user is completely unaware of the differences between an accessor and a virtual implementation.

[0250] A virtual implementation of the present invention takes a step toward the goal of a computer being able to modify its own programming. Normally, artificial intelligence is created in a neural network, genetic algorithm, or logic-based program. In every case, the computer is able to apply logic to "reason" out a result or create patterns of input that lead to patterns of output. While each of these programs allows the computer to construct new patterns, genes, or logic, none of these environments allow the computer to understand how to change the surrounding environment.